Università degli studi di Palermo 2019 - 2020

Raspberry Semaphore

CONTROLLO DI UN SEMAFORO ATTRAVERSO UN RASPBERRY PI 3B+ UTILIZZANDO FORTH

Giuseppe Terrasi | Sistemi Embedded | Prof. Daniele Peri

Sommario

1. Introduzione

Lo scopo di questo progetto è quello di costruire un semaforo a chiamata di un attraversamento pedonale e di controllarlo attraverso un Raspberry PI 3B+. Oltre al semaforo è presente anche uno display LCD 16x02 su cui viene visualizzato lo stato del semaforo ed il tempo rimanente prima che esso cambi.

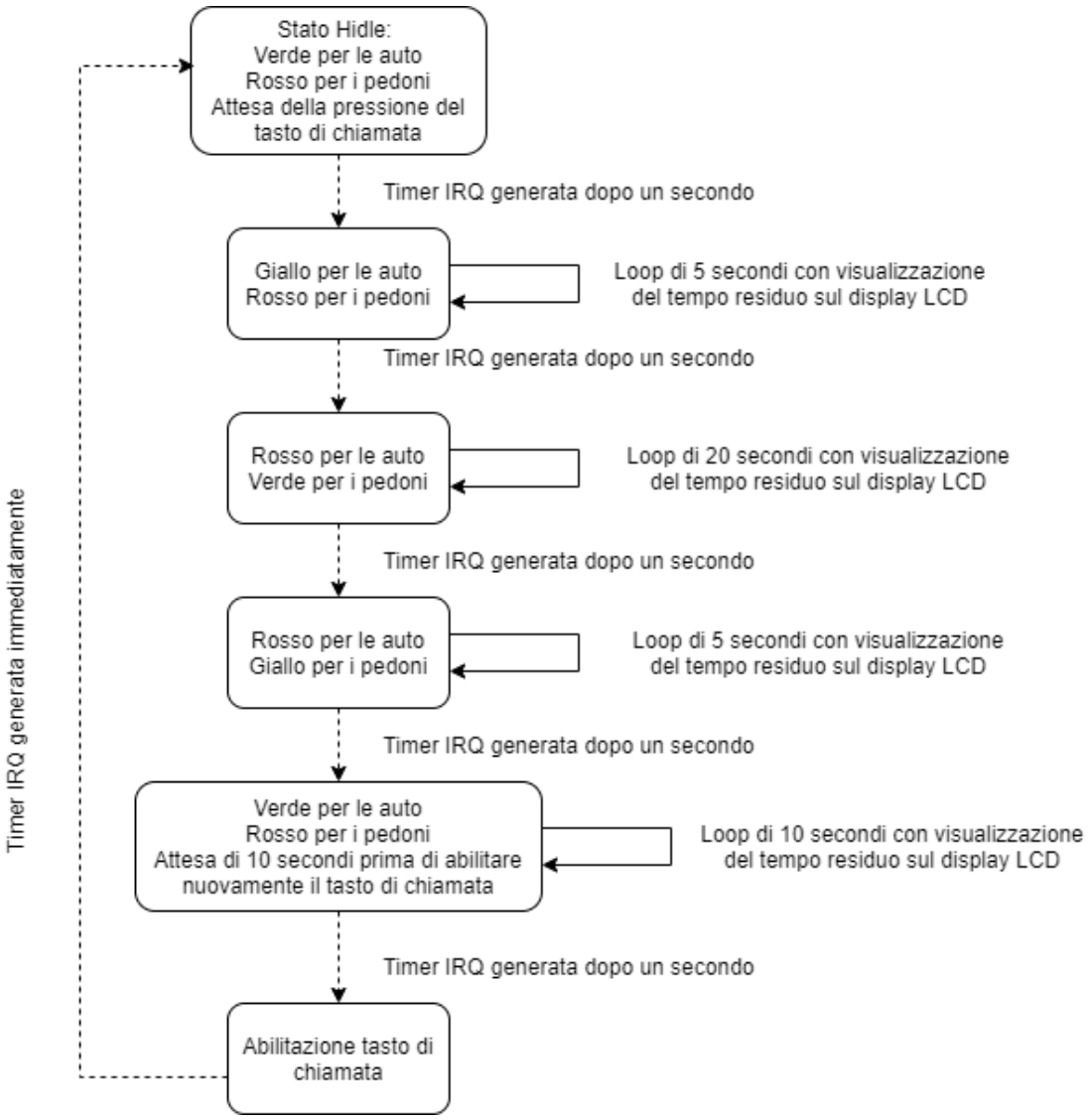
Lo stato iniziale del semaforo prevede Il verde per le auto, il rosso per i pedoni e la funzione di chiamata del rosso per le auto abilitata. Successivamente alla pressione del tasto di chiamata il semaforo inizia la routine di cambio stati di seguito elencati (per ragioni didattiche sono state usate delle tempistiche ridotte):

1. Spegnimento del verde per le auto ed accensione del giallo per esse. Attesa di cinque secondi.
2. Spegnimento del giallo per le auto ed accensione del rosso per esse con contestuale accensione del verde per i pedoni. Attesa di dieci secondi.
3. Spegnimento del verde per i pedoni ed accensione del giallo per essi. Attesa di cinque secondi.
4. Spegnimento del giallo per i pedoni ed accensione del rosso per essi con contestuale accensione del verde per le auto. Inibizione di un'ulteriore chiamata pedonale per dieci secondi.
5. Abilitazione della chiamata pedonale ed attesa indefinita.

Durante tutta la routine il display mostra sulla prima riga lo stato corrente del semaforo e sulla seconda riga il tempo rimanente per esso. Al punto 5 mostra solo sulla prima riga l'abilitazione della chiamata.

L'intero progetto è stato scritto nel linguaggio FORTH ed è eseguito in un ambiente installato sulla memoria SD del Raspberry PI 3B+.

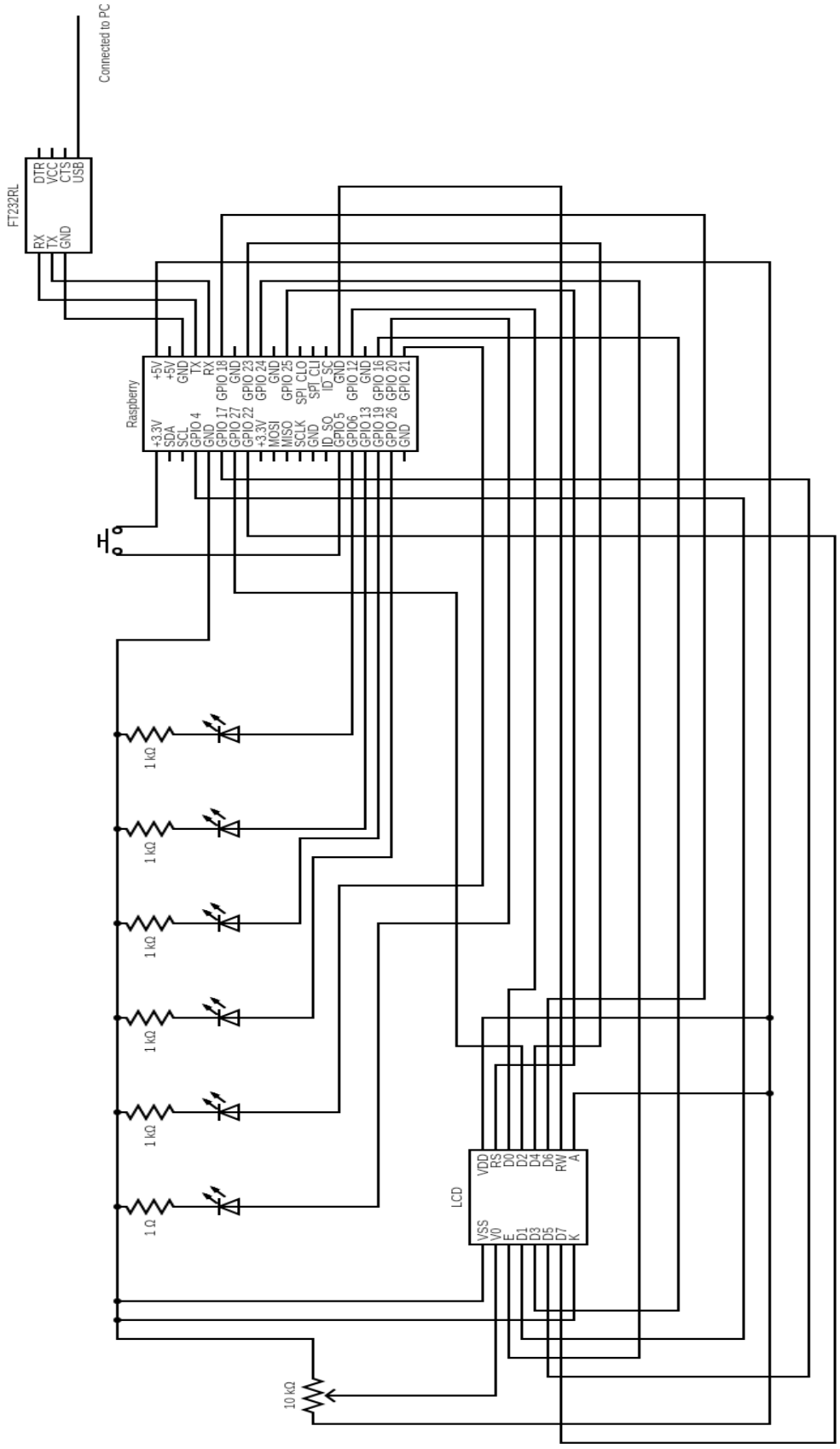
1.1. STATI DEL SEMAFORO



2. Componenti hardware utilizzati

- N. 1 Raspberry PI 3B+
- N. 1 Breadboard
- N. 1 LCD 16x02
- N. 2 LED Rossi
- N. 2 LED Gialli
- N. 2 LED Verdi
- N. 6 Resistenze da 1KΩ
- N. 1 Pulsante
- N. 1 Potenziometro da 10KΩ
- N. 1 GPIO Extension Board
- N. 1 40 pin GPIO Cable
- N. 1 FT232RL FTDI USB (Uart Serial)
- Jumper wires per i collegamenti

3. Circuito


Connected to PC
FT232RL
DTR
VCC
CTS
USB
RX
TX
GND
Raspberry
+5V
+5V
GND
TX
RX
GPIO 18
GND
GPIO 23
GPIO 24
GND
GPIO 25
SPI_CLO
SPI_CLI
GND
GPIO 12
GND
GPIO 16
GPIO 20
GPIO 21
+3.3V
SDA
SCL
GPIO 4
GND
GPIO 17
GPIO 27
GPIO 22
+3.3V
MOSI
MISO
SCLK
GND
ID_SO
GPIO6
GPIO 13
GPIO 19
GPIO 26
GND
1 kΩ
1 kΩ
1 kΩ
1 kΩ
1 kΩ
1 Ω
LCD
VDD
RS
D0
D2
D4
D6
RW
A
VSS
V0
E
D1
D3
D5
D7
K
10 kΩ

4. Descrizione dell'ambiente di lavoro

Repository GitHub: https://github.com/giuseppe-terrasi/raspberrypi-semaphore-forth

1. bootcode.bin → Raspberry 3B+ bootloader
2. fixup.dat → Linker file per il Raspberry
3. kernel.img → Immagine dell'interprete FORTH (pijforth modificato)
4. start.elf →Basic Raspberry Firmware
5. jonesforth.f → File del repository pjforth (https://github.com/organix/pijFORTHos/blob/master/jonesforth.f) per avere le word ." " – ABORT – s" " .
6. gpio.f → driver per l'utilizzo dei GPIO
7. time.f → driver per l'utilizzo del System Timer
8. lcd.f → driver per l'utilizzo del display LCD 16x02
9. semaphore.f → driver per controllare il semaforo
10. LICENSE → Licenza MIT del progetto
11. README.md → File di copertina per il repository GitHub

4.1. TOOL NECESSARI

- Sistema Operativo: **Windows 10** con il sottosistema Linux (**WSL**) abilitato e con la distribuzione **Debian** installata.
- IDE: **Visual Studio Code** con le estensioni per FORTH e Assembly
- Compilazione Assembly:
  - pacchetto **gcc-arm-none-eabi** installato sulla distribuzione Debian
  - software **make** installato sulla distribuzione Debian
- Interfacciamento con l'interprete FORTH sul Raspberry Pi: software **minicom** installato sulla distribuzione Debian con le seguenti impostazioni:
  - port: /dev/ttySN (il numero N è il numero della porta seriale assegnata da Windows quando si collega il dispositivo FT232RL FTDI tramite USB. È possibile ricavarlo dal pannello "Gestione dispositivi")
  - baudrate: 9600
  - bits: 8
  - parity: N
  - stopbits: 1
  - Terminale input delay: 2ms (utile quando si incollano diverse righe di codice)

N.B. il software minicom deve essere utilizzato con i privilegi di root

5. Descrizione drivers

5.1. NOTA IMPORTANTE

L'ordine con cui sono elencati i file in questo paragrafo corrisponde all'ordine in cui devono essere caricati i file sul Raspberry Pi.

5.2. INTERRUPT.F

Questo è il primo file da caricare in assoluto e deve essere necessariamente il primo in quanto contiene le istruzioni ARM eseguite dall'IRQ handler di basso livello. Per maggiori dettagli su come ricavare e posizionare queste istruzioni nella memoria consultare il paragrafo 5.2.8.

5.2.1. JF-HERE (--)

Mantiene il riferimento alla word HERE del kernel base.

5.2.2. HERE (-- addr)

Ridefinizione della word HERE in modo tale da lasciare sullo stack l'indirizzo dello stack pointer. Servirà principalmente per il calcolo dell'offset di salto per l'IRQ. (Paragrafo 5.2.8)

5.2.3. VARIABLE ITS

Variabile che contiene il timestamp in cui è verificata una IRQ. Viene valorizzata dall'IRQ Handler di basso livello e viene letta dal main loop per scatenare l'esecuzione dell'IRQ Handler di alto livello attraverso una vectored execution della word 'GENERIC_ISR. Tale handler viene settato nella variabile 'GENERIC_ISR dall'IRQ Handler di basso livello.

5.2.4. VARIABILE 'TIMER_ISR

Variabile che contiene l'indirizzo di memoria dell'IRQ Handler di alto livello per l'IRQ generata dal timer di sistema.

5.2.5. VARIABILE 'GPIO_ISR

Variabile che contiene l'indirizzo di memoria dell'IRQ Handler di alto livello per l'IRQ generata dal GPIO Async Fall Event.

5.2.6. VARIABLE 'GENERIC_ISR

Variabile che contiene l'indirizzo di memoria dell'IRQ Handler di alto livello da eseguire in base a quale IRQ è stata generata. Viene settata dall'IRQ Handler di basso livello e viene letta dal codice di alto livello, in particolare dal MAINLOOP.

5.2.7. CODICE MACCHINA IRQ DI BASSO LIVELLO

Dalla riga 16 alla riga 63 vengono caricate nella RAM le istruzioni eseguire dall'IRQ Handler di basso livello. Esse sono contenute nel file prodotto dal disassemblaggio del codice del file arm\interrupt.s. (Paragrafo 7)

5.2.8. IRQHANDLER! (--)

Word per settare all'indirizzo 0x18 l'istruzione di salto che punta all'indirizzo di memora di della prima istruzione del codice macchina del punto precedente.

L'istruzione di salto è un branch con un offset tra l'indirizzo della prima istruzione del dell'IRQ Handler caricato in precedenza e l'indirizzo 0x18, che è il

valore che contiene il PC (Program Counter) al momento della gestione dell'interruzione da parte del processore. Per sapere qual è l'indirizzo in questione basta eseguire la word HERE (Paragrafo 5.2.2) prima di caricare il codice macchina sulla RAM.

5.3. JONESFORTH.F

Questo file appartiene al repository originale da cui è stato ricavato il kernel utilizzato per il Raspberry Pi e che è possibile su GitHub in questo repository https://github.com/organix/pijFORTHos.

Il file è usato principalmente per la word s" utile per la creazione delle stringhe

5.4. GPIO.F

File contenente le word per poter utilizzare i GPIO del Raspberry Pi.

5.4.1. Costanti

GPFSEL0 = **0x3F200000** → Indirizzo del registro di controllo dei GPIO da 0 a 9 per la selezione della funzione del GPIO.

GPSET0 = 0x3F20001C → Indirizzo del registro per settare un GPIO da 0 a 30 con il valore "high"

GPCLR0 = 0x3F200028 → Indirizzo del registro per settare un GPIO da 0 a 30 con il valore "low"

GPLEV0 = 0x3F200034 → Indirizzo del registro contenenti i livelli dei GPIO da 0 a 30

GPAFEN0 = 0x3F200088 → Indirizzo del registro di controllo della rilevazione dell'aumento di tensione su un GPIO

IRQ2 = 0x 3F00B214 → Indirizzo del registro per l'abilitazione dell'IRQ line 2

5.4.2. GPIO (n - - n)

: **GPIO** DUP 30 > IF ABORT THEN ;

Questa word garantisce che il numero passato in input sia un valore valido per i GPIO. In questo progetto vengono usati i GPIO con numero inferiore a 30.

5.4.3. MODE (n -- a b c)

: **MODE** 10 /MOD 4 * GPFSEL0 + SWAP 3 * DUP 7 SWAP LSHIFT ROT DUP @ ROT INVERT AND ROT ;

Prende in input il numero del GPIO di cui impostare la modalità e lascia sullo stack il numero di left shift necessari per settare il corrispondente GPIO control bit del registro GPFSELN, dove N è il numero di registro, insieme all'indirizzo del registro GPFSELN ed il valore corrente settato in esso "pulito" da una maschera di bit. Il valore di N ed (a) sono calcolati dividendo il numero del

GPIO per 10; N sarà il quoziente moltiplicato per 4 mentre (a) sarà il resto. Il registro GPFSELN è calcolato a partire da GPFSEL0 aggiungendo il valore di N.

(es. GPIO 21 è controllato da GPFSEL2 quindi 21 / 10 --> N = 2 * 4, a = 1 --> GPFSEL0 + 8 = GPFSEL2)

I tre bit di controllo del registro GPFSELN che controllano il GPIO vengono impostati a zero tramite una machera di bit ottenuta shiftando verso sinistra il valore 0x7 di un numero di posizioni pari a 3 per il resto della divisione per 10 effettuata in precedenza.

(es. 21 / 10 -> 3 * 1 -> 7 shiftato a sinistra di 3)

5.4.4. OUTPUT (a b c --)

: **OUTPUT** 1 SWAP LSHIFT OR SWAP ! ;

Prende in input I tre valori lascati sullo stack dalla word MODE e setta la modalità output per il GPIO nel registro GPFSELN. Il bit del registro GPFSELN che controlla la modalità output del GPIO è settato tramite l'OR tra il valore memorizzato nel registro GPFSELN, "pulito" dalla maschera di bit, e un 1 shiftato a sinistra di un numero di posizioni pari al resto della divisione per 10 effettuata dalla word MODE moltiplicato per 3.

(es. Con GPIO 21 e @GPFSEL2: 011010--> 111000 011010 INVERT AND --> 000010 001000 OR --> 001010)

5.4.5. INPUT (a b c --)

: **INPUT** 1 SWAP LSHIFT INVERT AND SWAP ! ;

Prende in input I tre valori lascati sullo stack dalla word MODE e setta la modalità input per il GPIO nel registro GPFSELN.

Stesso meccanismo della word OUTPUT ma utilizza l'operazione INVERT AND invece dell'OR

5.4.6. ON (n --)

: **ON** 1 SWAP LSHIFT GPSET0 ! ;

Prende il numero del GPIO dallo stack, effettua un left shift del valore 1 di un numero di posizioni pari al numero del GPIO e setta il bit corrispondente del registro GPSET0.

5.4.7. OFF (n --)

: **OFF** 1 SWAP LSHIFT GPCLR0 ! ;

Prende il numero del GPIO dallo stack, effettua un left shift del valore 1 di un numero di posizioni pari al numero del GPIO e setta il bit corrispondente del registro GPCLR0.

5.4.8. LEVEL (n -- b)

: **LEVEL** 1 SWAP LSHIFT GPLEV0 @ SWAP AND ;

Prende il numero del GPIO dallo stack, effettua un left shift del valore 1 di un numero di posizioni pari al numero del GPIO, legge il valore del bit corrispondente del registro GPLEV0 e lo lascia sullo stack.

5.4.9. GPFSELOUT! (n --)

: **GPFSELOUT!** GPIO MODE OUTPUT ;

Scorciatoia per settate la modalità output di un GPIO

5.4.10. GPFSELIN! (n --)

: **GPFSELIN!** GPIO MODE INPUT ;

Scorciatoia per settate la modalità input di un GPIO

5.4.11. GPON! (n --)

: **GPON!** GPIO ON ;

Scorciatoia per settare un GPIO con il valore “high”.

5.4.12. GPOFF! (n --)

: **GPOFF!** GPIO OFF ;

Scorciatoia per settare un GPIO con il valore “low”.

5.4.13. GPLEV@ (n – n1)

: **GPLEV@** GPIO LEVEL ;

Scorciatoia per leggere il valore del GPIO passato in input.

5.4.14. GPAFEN! (n --)

: **GPAFEN!** GPIO 1 SWAP LSHIFT GPAFEN0 ! ;

Abilita la fall detection asincrona del livello del GPIO n-esimo.

5.4.15. IRQGPIO! (--)

: **IRQGPIO!** 1 17 LSHIFT IRQ2 @ OR IRQ2 ! ;

Abilita la sorgente IRQ per le interruzioni generate dai GPIO da 1 a 30.

5.5. TIME.F

File contenente le word per poter utilizzare le funzionalità del timer di sistema.

N.B. Il timer ha 4 comparatori C0, C1, C2 e C3 ma le applicazioni possono utilizzarne solo 2, C1 e C3, poiché C0 e C2 vengono utilizzati dalla GPU.

5.5.1. Costanti

SYSCS = **0x3F003000** → Registro di controllo / stato del System Timer

SYSCLO = **0x3F003004** → Registro contenente i 32 bit inferiori del System Timer Counter

SYSCHI = **0x3F003008** → Registro contenente i 32 bit superiori del System Timer Counter

SYSC1 = **0x3F003010** → Indirizzo del registro del System Timer Compare 1

SYSC3 = **0x3F003018** →Indirizzo del registro del System Timer Compare 3

IRQ1 = 0x3F00B210 → Indirizzo del registro per l'abilitazione dell'IRQ line 1

5.5.2. NOW (-- n)

: **NOW** SYSCLO @ ;

Legge il valore attuale dei 32 bit inferiori del System Timer Counter corrispondenti al valore in microsecondi dall'avvio del sistema (il valore esatto è l'intero valore a 64 bit contenuto in SYSCLO e SYSCHI. In questo progetto è sufficiente leggere il solo valore di SYSCLO)

5.5.3. USEC

: **USEC** ;

Unità di misura per i microsecondi equivalente alla moltiplicazione per 1 in quanto il timer di sistema viene aggiornato ogni microsecondo.

5.5.4. MSEC

: **MSEC** 1000 * ;

Unità di misura per i millisecondi, mille microsecondi.

5.5.5. SEC

: **SEC** 1000 MSEC * ;

Unità di misura per i secondi, mille millisecondi.

5.5.6. MIN

: **MIN** 60 SEC * ;

Unità di misura per i minuti, 60 secondi.

5.5.7. DELAY (n --)

: **DELAY** NOW + BEGIN DUP NOW - 0 <= UNTIL DROP ;

Word per eseguire un delay di un tempo pari ad n microsecondi utilizzando un busy wait.

5.5.8. SYSC1! (n --)

: **SYSC1!** NOW + SYSC1 ! ;

Word per impostare il System Timer Compare 1 con un valore pari a NOW + n. Il timer confronta il valore corrente con il valore contenuto in questo registro e quando combacia lancia un'interruzione IRQ sulla linea 1.

5.5.9. SYSC3! (n --)

: **SYSC3!** NOW + SYSC3 ! ;

Word per impostare il System Timer Compare 3 con un valore pari a NOW + n. Il timer confronta il valore corrente con il valore contenuto in questo registro e quando combacia lancia un'interruzione IRQ sulla linea 1.

5.5.10. IRQTIMER1! (--)

: **IRQTIMER1!** IRQ1 @ 2 OR IRQ1 ! ;

Abilita la sorgente IRQ per le interruzioni generate dal timer di sistema.

5.6. LCD.F

File contenente le word per poter utilizzare l'LCD 16x02. Per questo progetto è stata utilizzata la configurazione a 8 bit.

5.6.1. Costanti

LCDRS = 25 → GPIO pin a cui è connesso il pin RS del display LCD

LCDE = 24 → GPIO pin a cui è connesso il pin E del display LCD

LCD0 = 12 → GPIO pin a cui è connesso il pin dati 0 del display LCD

LCD1 = 4 → GPIO pin a cui è connesso il pin dati 1 del display LCD

LCD2 = 27 → GPIO pin a cui è connesso il pin dati 2 del display LCD

LCD3 = 16 → GPIO pin a cui è connesso il pin dati 3 del display LCD

LCD4 = 23 → GPIO pin a cui è connesso il pin dati 4 del display LCD

LCD5 = 17 → GPIO pin a cui è connesso il pin dati 5 del display LCD

LCD6 = 18 → GPIO pin a cui è connesso il pin dati 6 del display LCD

LCD7 = 22 → GPIO pin a cui è connesso il pin dati 7 del display LCD

LCDLN1 = 0x80 → Indirizzo di memoria della DDRAM del display LCD per la linea 1

LCDLN2 = 0xC0 → Indirizzo di memoria della DDRAM del display LCD per la linea 2

5.6.2. LCDSETUP (--)

```
: LCDSETUP
  LCDE  GPFSELOUT!
  LCDRS GPFSELOUT!
  LCD0 GPFSELOUT!
  LCD1 GPFSELOUT!
  LCD2 GPFSELOUT!
  LCD3 GPFSELOUT!
  LCD4 GPFSELOUT!
  LCD5 GPFSELOUT!
  LCD6 GPFSELOUT!
  LCD7 GPFSELOUT! ;
```

Imposta la funzione output per tutti i GPIO pin connessi ai pin del display LCD.

5.6.3. +LCDE (--)

: **+LCDE** LCDE GPON! ;

Shortcut per settare in 'alto' l'output del GPIO pin a cui è connesso il pin E del display LCD.

5.6.4. -LCDE (--)

: **-LCDE** LCDE GPOFF! ;

Shortcut per settare in 'basso' l'output del GPIO pin a cui è connesso il pin E del display LCD.

5.6.5. LCDESIGN (--)

: **LCDESIGN** 500 USEC DELAY +LCDE 500 USEC DELAY -LCDE 500 USEC DELAY ;

Invia un segnale alto-basso a pin E del display LCD.

5.6.6. LCDDOFF! (--)

: **LCDDOFF!**

LCDE GPOFF!

LCD0 GPOFF!

LCD1 GPOFF!

LCD2 GPOFF!

LCD3 GPOFF!

LCD4 GPOFF!

LCD5 GPOFF!

LCD6 GPOFF!

LCD7 GPOFF!

;

Imposta in 'basso' l'output dei pin GPIO a cui sono connessi il pin E ed i pin dati del display LCD

5.6.7. BITSET (n1 n2 -- flag)

: **BITSET** AND 0<> ;

Controlla se il bit rappresentato da n2 è settato in n1 e lascia sullo stack il flag corrispondente. Per poter essere usata correttamente n2 deve essere un 1 shiftato a sinistra di x posizioni corrispondenti alla posizione del bit da testare.

5.6.8. ?LCDD (flag n --)

: **?LCDD** SWAP IF GPON! ELSE GPOFF! THEN ;

Accende o spegne il GPIO n in base al valore del flag.

5.6.9. LCDWRITE (n --)

```
: LCDWRITE
   DUP 100 BITSET LCDRS ?LCDD
   DUP 80 BITSET LCD7 ?LCDD
   DUP 40 BITSET LCD6 ?LCDD
   DUP 20 BITSET LCD5 ?LCDD
   DUP 10 BITSET LCD4 ?LCDD
   DUP 08 BITSET LCD3 ?LCDD
   DUP 04 BITSET LCD2 ?LCDD
   DUP 02 BITSET LCD1 ?LCDD
      01 BITSET LCD0 ?LCDD
   LCDESIGN
;
```

Invia il valore di n al display LCD in modalità 8 bit. Se n < 0x100 corrisponde ad un comando altrimenti corrisponde a dei dati.

5.6.10. LCDWRITE4 (n --)

```
: LCDWRITE4
   DUP 100 BITSET LCDRS ?LCDD
   \ set high bits
   DUP 80 BITSET LCD7 ?LCDD
   DUP 40 BITSET LCD6 ?LCDD
   DUP 20 BITSET LCD5 ?LCDD
   DUP 10 BITSET LCD4 ?LCDD
   LCDESIGN
   \ set low bits
   DUP 08 BITSET LCD7 ?LCDD
   DUP 04 BITSET LCD6 ?LCDD
   DUP 02 BITSET LCD5 ?LCDD
      01 BITSET LCD4 ?LCDD
   LCDESIGN
;
```

Invia il valore di n al display LCD in modalità 4 bit. Se n < 0x100 corrisponde ad un comando altrimenti corrisponde a dei dati.

5.6.11. LCDCLR (--)

```
: LCDCLR 1 LCDWRITE ;
```

Invia il comando 0x1 al display LCD che pulisce tutto lo schermo.

5.6.12. LCDINIT (--)

```
: LCDINIT
   33 LCDWRITE
   38 LCDWRITE
   6 LCDWRITE
   C LCDWRITE
   LCDCLR
;
```

Word che inizializza il display LCD impostando anche la modalità 8 bit.

5.6.13. LCDSTYPE (addr n --)

: **LCDSTYPE** OVER + SWAP BEGIN 2DUP <> WHILE DUP c@ 100 + LCDWRITE 1+ REPEAT 2DROP ;

Visualizza sul display la stringa contenuta tra l'indirizzo di memoria addr e addr + n. La stringa comprende anche il carattere \0 di chiusura.

5.6.14. LCDSTYPE (n --)

: **LCDNTYPE** BEGIN 10 /MOD SWAP 304 + LCDWRITE DUP 0= UNTIL DROP ;

Visualizza sul display la rappresentazione in stringa del numero n.

5.6.15. LCDLN (n -- addr)

: **LCDLN** 1 = IF LCDLN1 ELSE LCDLN2 THEN ;

Lascia sullo stack l'indirizzo di memoria della riga n del display LCD nella DDRAM.

5.6.16. LCDLN (n -- addr)

: **LCDLN!** LCDLN + LCDWRITE ;

Recupera l'indirizzo nella DDRAM della riga n del display LCD da dove il cursore inizierà a visualizzare i caratteri e lo invia al display.

5.6.17. LCDCURL>R (--)

: **LCDCURL>R** 6 LCDWRITE ;

Invia al display LCD il comando per impostare il movimento del cursore da sinistra a destra.

5.6.18. LCDCURL<R (--)

: **LCDCURL<R** 4 LCDWRITE ;

Invia al display LCD il comando per impostare il movimento del cursore da destra a sinistra.

5.6.19. LCDSTRING (addr n1 n2 n3 --)

: **LCDSTRING** LCDLN! LCDCURL>R LCDSTYPE ;

Visualizza sul display, alla riga corrispondente al valore di n3, la stringa contenuta tra l'indirizzo di memoria addr e addr + n1 da un offeset di n2 dall'inizio della riga n3. La stringa viene scritta da sinistra a destra.

5.6.20. LCDNUMBER (n1 n2 n3 --)

: **LCDNUMBER** LCDLN! LCDCURL<R LCDNTYPE ;

Visualizza sul display, alla riga corrispondente al valore di n3, la rappresentazione in stringa del numero n1 da un offeset di n2 dall'inizio della riga n3. La stringa viene scritta da destra a sinistra poiché il numero n1 viene convertito dalla cifra più bassa a quella più alta.

5.6.21. LCDSPACE (--)

: **LCDSPACE** 120 LCDWRITE ;

Visualizza sul display uno spazio.

5.6.22. LCDLNCLR (n --)

: **LCDLNCLR** 0 SWAP LCDLN! 16 0 BEGIN LCDSPACE 1 + 2DUP = UNTIL 2DROP ;

Pulisce la riga n del display LCD.

5.7. SEMAPHORE.F

File contenete le word per poter utilizzare il semaforo.

5.7.1. Costanti

RCAR = 20 → GPIO pin a cui è connesso il led rosso del semaforo per le auto

YCAR = 21 → GPIO pin a cui è connesso il led giallo del semaforo per le auto

GCAR = 26 → GPIO pin a cui è connesso il led verde del semaforo per le auto

RPED = 19 → GPIO pin a cui è connesso il led rosso del semaforo per i pedoni

YPED = 13 → GPIO pin a cui è connesso il led giallo del semaforo per i pedoni

GPED = 6 → GPIO pin a cui è connesso il led verde del semaforo per i pedoni

BUTTON = 5 → GPIO pin a cui è connesso il pulsante di chiamata. Questo GPIO è in modalità INPUT.

5.7.2. RED (-- n1 n2)

: **RED** RCAR RPED ;

Lascia sullo stack il numero dei GPIO pin dei led rossi.

5.7.3. YELLOW (--)

: **YELLOW** YCAR YPED ;

Lascia sullo stack il numero dei GPIO pin dei led gialli.

5.7.4. GREEN (--)

: **GREEN** GCAR GPED ;

Lascia sullo stack il numero dei GPIO pin dei led verdi.

5.7.5. CAR (-- n)

: **CAR** DROP ;

Rimuove dallo stack il numero del GPIO pin del dei pedoni. Deve essere utilizzata subito dopo una della word RED, YELLOW e GREEN.

5.7.6. PEDESTRIAN (--)

: **PEDESTRIAN** SWAP DROP ;

Rimuove dallo stack il numero del GPIO pin delle auto. Deve essere utilizzata subito dopo una della word RED, YELLOW e GREEN.

5.7.7. LCDSETUP (--)

: **LCDSETUP**

```
LCDE  GPFSELOUT!
LCDRS GPFSELOUT!
LCD0 GPFSELOUT!
LCD1 GPFSELOUT!
LCD2 GPFSELOUT!
LCD3 GPFSELOUT!
LCD4 GPFSELOUT!
LCD5 GPFSELOUT!
LCD6 GPFSELOUT!
LCD7 GPFSELOUT!
```

;

Imposta la funzione output per tutti i GPIO pin connessi ai led del semaforo.

5.7.8. SEMINIT (--)

: **SEMINIT**

```
RED CAR GPOFF!
YELLOW CAR GPOFF!
GREEN CAR GPON!
RED PEDESTRIAN GPON!
YELLOW PEDESTRIAN GPOFF!
GREEN PEDESTRIAN GPOFF!
```

;

Word che imposta lo stato iniziale dei led del semaforo, in particolare verde per le auto e rosso per i pedoni.

5.8. MAIN.F

Questo file contiene sia la definizione delle word relative alla logica di funzionamento di tutti i componenti e di come devono susseguirsi i vari stati del semaforo. Per maggiori dettagli consultare il paragrafo 1.1.

5.8.1. Variabili

LASTRUN → Variabile utilizzata per tenere traccia dell'istante in cui è stato eseguito l'ultima volta l'IRQ Handler di alto livello. Viene comparata con la variabiel ITS e quando le due sono differenti, ovvero quando è stata lanciata una nuova interruzione IRQ, viene eseguita l'azione successiva nella sequenza di azioni del semaforo.

STARTTIME → Variabile utilizzata per tenere traccia dell'istante in cui è iniziato il main loop del funzionamento del semaforo. È utilizzata solo a scopo di debug.

ACTION_IDX → Variabile contenente l'indice della successiva azione da eseguire.

ACTION_LEN → Variabile contenente la durata dell'azione dell'indice ACTION_IDX.

FULLSEMLEN → Variabile contenente la durata di un'intera routine del semaforo. Ce ne sono 2, quella dalla pressione del tasto di chiamata a quando il semaforo torna ad essere rosso per i pedoni e quella di inibizione della chiamata pedonale per X secondi.

ACTIONS → Variabile di tipo array contenente l'elenco ordinato delle azioni da seguite. Tramite il valore di ACTION_IDX è possibile stabilire la prossima azione da eseguire.
La variabile è valorizzata con l'indirizzo di memoria da cui inizia l'array e che viene allocato tramite l'istruzione:

```
6 CELLS ALLOT ACTIONS !
```

La word ALLOT lascia sullo stack l'indirizzo di memoria utile da assegnare alla variabile.

5.8.2. ACTION (n – addr)

```
: ACTION CELLS ACTIONS @ + ;
```

Restituisce e lascia sullo stack l'indirizzo di memoria dell'azione all'indice n nell'array ACTIONS.

5.8.3. FULLSEMLEN-- (--)

```
: FULLSEMLEN-- 1 FULLSEMLEN -! ;
```

Decrementa di 1 il valore della variabile FULLSEMLEN.

5.8.4. ACTION_IDX+ (n --)

: **ACTION_IDX+** ACTION_IDX +! ;

Incrementa di n il valore dell'indice ACTION_IDX.

5.8.5. ACTION_LEN! (n --)

: **ACTION_LEN!** ACTION_LEN ! ;

Setta la variabile ACTION_LEN con il valore n

5.8.6. ACTION_LEN-- (--)

: **ACTION_LEN--** 1 ACTION_LEN -! ;

Decrementa di 1 il valore della variabile ACTION_LEN.

5.8.7. ISCHED! (--)

: **ISCHED!** 1 SEC SYSC1! ;

Schedula tra 1 secondo la prossima interruzione IRQ generata dal timer di sistema.

5.8.8. NEXTACTION! (n m --)

: NEXTACTION! ACTION_IDX+ ACTION_LEN! ISCHED! ;

Setta le variabili necessarie ad eseguire la prossima azione nella sequenza. In particolare, incrementa la variabile ACTION_IDX con il valore n (non è stata usata una word che incrementa automaticamente di 1 perché l'ultima azione nella sequenza passa in n un valore negativo utile a riportare a 0 il valore di ACTION_IDX), imposta la durata dell'azione nella variabile ACTION_LEN con il valore m e schedula la prossima interruzione del timer di sistema. Ogni azione della sequenza utilizza questa word come ultima istruzione innescando il meccanismo di cambio stato del semaforo. Fa eccezione l'azione numero 0 in quanto è l'azione corrispondente allo stato hidle del semaforo (sempre verde per le auto). Ciò che genera il cambio stato in questo caso è la pressione del tasto di chiamata. (Paragrafo 5.8.22).

5.8.9. 0ACTION (--)

: **0ACTION** LCDCLR S" P. RED CAN CALL " 0 1 LCDSTRING 30 FULLSEMLEN ! ;

Azione corrispondente allo stato hidle del semaforo. Visualizza sulla prima riga del display il messaggio "P. RED CAN CALL" ovvero rosso per i pedoni, chiamata abilitata. Infine, setta a 30 secondi la durata della prima routine del semaforo il cui inizio è scatenato dalla pressione del tasto di chiamata pedonale.

5.8.10. 1ACTION (--)

: **1ACTION** S" CAR YELLOW " 0 1 LCDSTRING

GREEN CAR GPOFF! YELLOW CAR GPON!

5 1 NEXTACTION! ;

Questa è la prima azione che viene eseguita successivamente alla pressione del tasto di chiamata. Visualizza sulla prima riga del display il messaggio "CAR YELLOW " (gli spazi in più servono a cancellare i caratteri non necessari del messaggio precedente) ovvero giallo per le auto, spegne il led verde per le auto e accende il led giallo per esse, incrementa l'indice ACTION_IDX di 1, setta i secondi per i quali il semaforo rimarrà in questo stato (5 secondi) e infine schedula tra un secondo una interruzione IRQ generata dal timer di sistema.

5.8.11. 2ACTION (--)

: **2ACTION** S" P. GREEN " 0 1 LCDSTRING

RED CAR GPON! YELLOW CAR GPOFF! RED PEDESTRIAN GPOFF! GREEN PEDESTRIAN GPON!

19 1 NEXTACTION! ;

Visualizza sulla prima riga del display il messaggio " P. GREEN " (gli spazi in più servono a cancellare i caratteri non necessari del messaggio precedente) ovvero verde per i pedoni; spegne il led giallo per le auto, accende il led rosso per esse, spegne il led rosso per i pedoni ed accende il led verde per essi; incrementa l'indice ACTION_IDX di 1, setta i secondi per i quali il semaforo rimarrà in questo stato (20 secondi, ma ne vengono settati 19 poiché l'azione stessa schedula la prima interruzione dopo 1 secondo) e infine schedula tra un secondo una interruzione IRQ generata dal timer di sistema.

5.8.12. 3ACTION (--)

: **3ACTION** S" P. YELLOW " 0 1 LCDSTRING

GREEN PEDESTRIAN GPOFF! YELLOW PEDESTRIAN GPON!

4 1 NEXTACTION! ;

Visualizza sulla prima riga del display il messaggio "P. YELLOW" ovvero giallo per i pedoni; spegne il led verde per i pedoni ed accende il led giallo per essi; incrementa l'indice ACTION_IDX di 1, setta i secondi per i quali il semaforo rimarrà in questo stato (5 secondi, ma ne vengono settati 4 poiché l'azione stessa schedula la prima interruzione dopo 1 secondo) e infine schedula tra un secondo una interruzione IRQ generata dal timer di sistema.

5.8.13. 4ACTION

: **4ACTION** S" P. RED WAIT " 0 1 LCDSTRING

RED PEDESTRIAN GPON! YELLOW PEDESTRIAN GPOFF! RED CAR OFF GREEN CAR ON

10 FULLSEMLEN !

9 1 NEXTACTION! ;

- Visualizza sulla prima riga del display il messaggio " P. RED WAIT " ovvero rosso per i pedoni e attesa prima della prossima attivazione del tasto di chiamata;
- spegne il led giallo per i pedoni ed accende il led rosso per essi, spegne il led rosso per le auto ed accende quello verde per essi;
- setta a dieci secondi la durata della seconda routine del semaforo, ovvero la disabilitazione del tasto di chiamata. In questo modo sulla seconda riga il timer ricomincerà da 10.
- incrementa l'indice ACTION_IDX di 1, setta i secondi per i quali il semaforo rimarrà in questo stato (10 secondi, ma ne vengono settati 9 poiché l'azione stessa schedula la prima interruzione dopo 1 secondo) e infine schedula tra un secondo una interruzione IRQ generata dal timer di sistema.

5.8.14. 5ACTION (--)

: **5ACTION** 0 -5 NEXTACTION! ;

Questa azione riporta il semaforo nello stato hidle. La sua durata è nulla e decrementa l'indice ACTION_IDX di 5 riportandolo quindi a 0. In questo modo inoltre viene riabilitato il tasto di chiamata in quanto l'IRQ Handler di alto livello per esso controlla il valore dell'indice e solo quando quest'ultimo è a 0 esegue la word 1ACTION.

5.8.15. EXEC_NEXT (--)

: **EXEC_NEXT** ACTION_IDX @ ACTION @ EXECUTE ;

Utilizzando il pattern vectored execution, esegue l'azione corrispondente al valore dell'indice ACTION_IDX (Es. ACTION_IDX 0 → 0ACTION). L'indirizzo di memoria della word che esegue l'azione da eseguire è memorizzato nell'indice ACTION_IDX dell'array ACTIONS e, una volta recuperato, la word in questione viene eseguite tramite la word EXECUTE.

5.8.16. SETn (--)

Queste word servono a settare nell'indice n dell'array ACTIONS l'indirizzo di memoria della word n corrispondente all'azione n del semaforo.

- : SET0 ' 0ACTION 0 ACTION ! ;
- : SET1 ' 1ACTION 1 ACTION ! ;

- : SET2 ' 2ACTION 2 ACTION ! ;
- : SET3 ' 3ACTION 3 ACTION ! ;
- : SET4 ' 4ACTION 4 ACTION ! ;
- : SET5 ' 4ACTION 5 ACTION ! ;

L'indirizzo di memoria della word n viene recuperato tramite la word ' (tick). Dopodiché tramite la word ACTION viene recuperato l'indirizzo di memoria all'interno dell'array ACTIONS in cui memorizzare la word n.

5.8.17. TIMER_ISR (--) – IRQ Handler di alto livello per le IRQ del timer di sistema

: **TIMER_ISR**

FULLSEMLEN @ 10 < IF 1 2 LCDLN! LCDSPACE THEN

FULLSEMLEN @ 2 2 LCDNUMBER

ACTION_LEN @

0= IF EXEC_NEXT

ELSE ACTION_LEN-- ISCHED! THEN

FULLSEMLEN-- ;

- Visualizza sulla seconda riga del display LCD il valore corrente del tempo rimanente per la routine in corso. Se il valore è inferiore a 10 aggiunge uno spazio all'inizio in modo tale da cancellare la cifra delle decine.
- Se l'azione corrente è terminata (ACTION_LEN = 0) esegue la prossima azione nella sequenza tramite la word EXEC_NEXT altrimenti decrementa di uno il valore ACTION_LEN, segnalando che è passato un secondo e schedula la prossima IRQ tra un secondo (ed in questo modo anche la sua stessa esecuzione)
- Decrementa di uno il valore della durata totale della routine. In questo modo alla prossima esecuzione di **TIMER_ISR** verrà visualizzata l'informazione che è passato un secondo.

5.8.18. GPIO_ISR (--) – IRQ Handler di alto livello per le IRQ dei GPIO

: **GPIO_ISR** ACTION_IDX @ 0= IF 1 ACTION_IDX+ EXEC_NEXT THEN ;

Controlla che l'azione corrente del semaforo sia quella di hidle ovvero ACTION_IDX = 0 e solo in questo caso incrementa l'indice ed avvia la sequenza.

5.8.19. TIMER_ISR! (--)

: **TIMER_ISR!** ' TIMER_ISR 'TIMER_ISR ! ;

Setta l'indirizzo di memoria della word TIMER_ISR nella variabile 'TIMER_ISR letta dall'IRQ Handler di basso livello.

5.8.20. GPIO_ISR! (--)

: **GPIO_ISR!** ' GPIO_ISR 'GPIO_ISR ! ;

Setta l'indirizzo di memoria della word GPIO_ISR nella variabile 'GPIO_ISR letta dall'IRQ Handler di basso livello.

5.8.21. INIT_ALL (--)

: **INIT_ALL**

SET0

SET1

SET2

SET3

SET4

SET5

TIMER_ISR!

GPIO_ISR!

IRQHANDLER!

SEMSETUP

LCDSETUP

SEMINIT

LCDINIT

0 ACTION_IDX !

30 FULLSEMLEN !

5 GPAFEN!

IRQGPIO!

IRQTIMER1!

+IRQ ;

Word che inizializza tutti i componenti per il corretto funzionamento di tutto il sistema.

- Valorizza l'array ACTIONS
- Valorizza le variabili contenenti gli indirizzi di memoria degli IRQ Handler di alto livello

- Setta all'indirizzo di memoria 0x18 (letto dal processore quando si verifica una IRQ) l'istruzione di salto all'indirizzo di memoria dell'IRQ Handler di basso livello
- Inizializza i GPIO collegati ai led del semaforo ed imposta lo stato iniziale
- Inizializza i GPIO collegati al display LCD e successivamente inizializza il display stesso
- Inizializza l'indice ACTION_IDX a 0 corrispondente allo stato hidle del semaforo
- Imposta l'evento GPIO Async Fall Event per il GPIO 5 collegato al bottone per la chiamata pedonale
- Abilita la sorgente IRQ per i GPIO
- Abilita la sorgente IRQ per il timer di sistema
- Abilita la gestione delle IRQ da parte del processore

5.8.22. MAINLOOP (--)

: **MAINLOOP**

NOW STARTTIME !

STARTTIME @ ITS !

STARTTIME @ LASTRUN !

EXEC_NEXT

BEGIN

NOW STARTTIME @ - 1 MIN < WHILE

LASTRUN @ ITS @ <> IF ITS @ LASTRUN ! 'GENERIC_ISR @ EXECUTE THEN

REPEAT ;

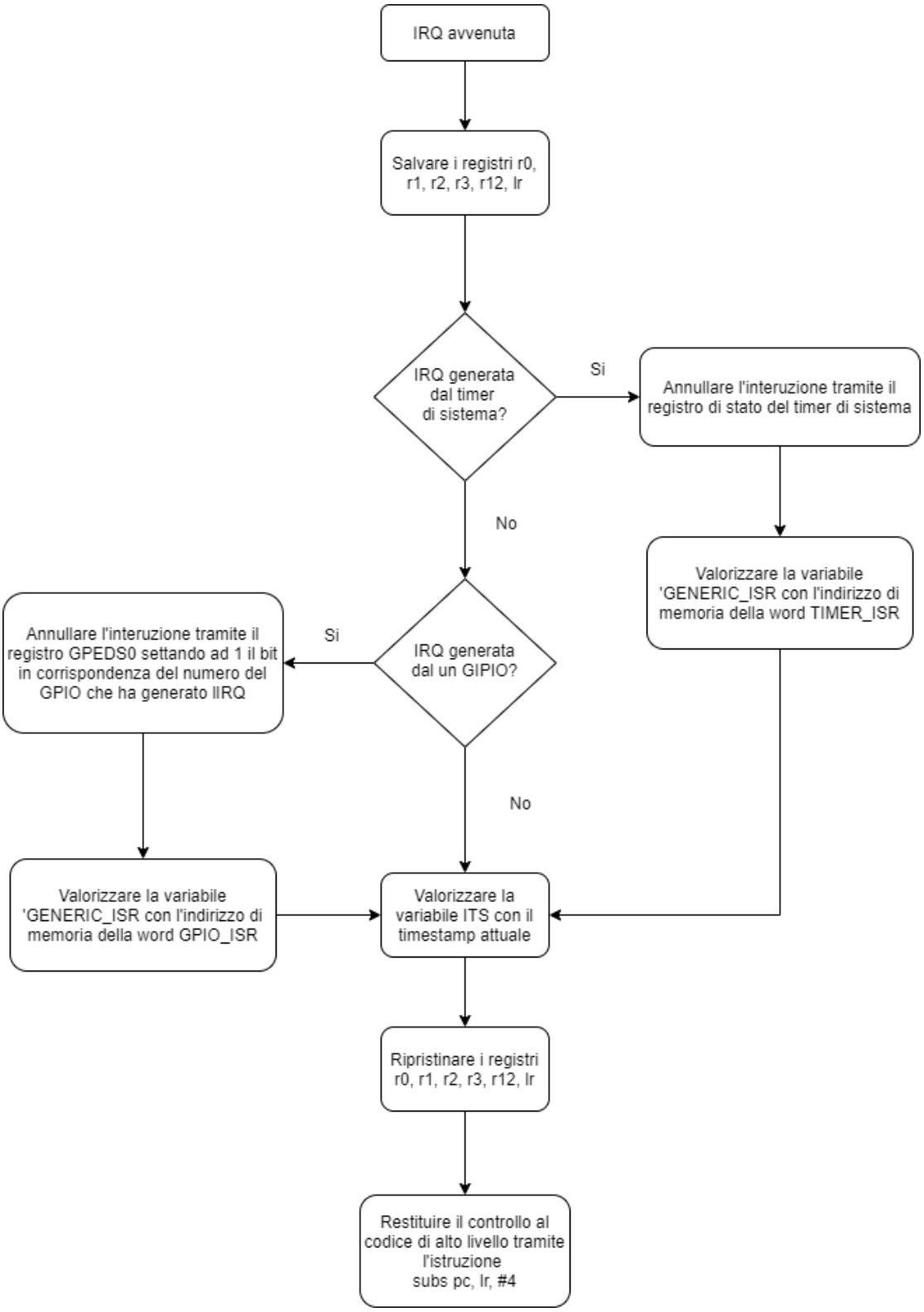
N.B la word deve essere eseguita dopo la word INIT_ALL.

Loop che permette l'esecuzione di un IRQ Handler di alto livello successivamente all'esecuzione di uno di basso livello.

Il loop dovrebbe essere infinito ma per poter debuggare il software è finito e dura un minuto.

La word setta le variabili per uscire dal loop dopo un minuto ed esegue EXEC_NEXT per settare le variabili relative all'azione 1.
Successivamente controlla periodicamente se il processore abbia gestito una IRQ confrontando le variabili **LASTRUN** (aggiornata quando viene eseguito un IRQ Handler di alto livello) e **ITS** (aggiornata dall'IRQ Handler di basso livello); quando sono diverse, tramite il pattern vectored execution, esegue l'IRQ Handler di alto livello il cui indirizzo di memoria è stato settato nella variabile 'GENERIC_ISR dall'IRQ Handler di basso livello.

6. Gestione delle interruzioni



Il grafico in figura mostra le operazioni eseguite quando viene generata un’interruzione IRQ.

7. Compilare e disassemblare il codice macchina per l'IRQ di basso livello

L'IRQ Handler è scritto in codice assembly e per poter essere utilizzato all'interno dell'ambiente FORTH è necessario compilarlo e disassemblare il file binario prodotto dalla compilazione, ottenendo le istruzioni macchina codificate in numeri esadecimali. Queste istruzioni possono essere memorizzate nella RAM con la word FORTH , (comma) ad esempio:

e92d500f ,

L'intero codice assembly dell'handler è contenuto nel file asm\interrupt.s.

È necessario anche conoscere l'istruzione di salto codificata da memorizzare all'indirizzo 0x18 che è l'indirizzo che viene letto dal processore quando si verifica una interruzione IRQ. Il codice assembly per ottenere questa istruzione è contenuto nel file asm\sethandlerjump.s

Per ottenere le istruzioni macchina, nella carella asm è presente un **Makefile** il quale, tramite il tool **make**, compila i file **.s** e genera i file **elf.list** contenenti il codice macchina. Affinché la compilazione avvenga con successo è necessario che sul sistema siano installati i GNU ARM Embedded Toolchain. (Paragrafo 4.1)